Lecture 14

Greedy Method: Fractional Knapsack, Interval scheduling

CS 161 Design and Analysis of Algorithms

Ioannis Panageas

Greedy method

The greedy method is a general algorithm design technique, in which given:

- configurations: different choices we need to make
- objective function: a score assigned to all configurations, which we want to either maximize or minimize

We should make choices greedily: We can find a globally-optimal solution by a series of local improvements from a starting configuration.

Greedy method

The greedy method is a general algorithm design technique, in which given:

- configurations: different choices we need to make
- objective function: a score assigned to all configurations, which we want to either maximize or minimize

We should make choices greedily: We can find a globally-optimal solution by a series of local improvements from a starting configuration.

Example: Maxflow problem.

Configurations: All possible flow functions. Objective function: Maximize flow value.

Ford-Fulkerson makes choices greedily starting from flow $f = 0$.

Greedy does not always work

Problem 1: Given a value X and notes $\{1, 2, 5, 10, 20, 50, 100\}$, find the minimum number of notes to create value X. You can use each note as many times as you want.

Greedy does not always work

Problem 1: Given a value X and notes $\{1, 2, 5, 10, 20, 50, 100\}$, find the minimum number of notes to create value X. You can use each note as many times as you want.

Answer: Greedy approach works. Pick largest note that is at most X and subtract from X. Repeat until value becomes 0.
E.g., for X=1477, you need fourteen 100s, one 50, one 20, one 5 and one 2.

Greedy does not always work

Problem 1: Given a value X and notes $\{1, 2, 5, 10, 20, 50, 100\}$, find the minimum number of notes to create value X. You can use each note as many times as you want.

Answer: Greedy approach works. Pick largest note that is at most X and subtract from X. Repeat until value becomes 0.
E.g., for X=1477, you need fourteen 100s, one 50, one 20, one 5 and one 2.

Problem 2: Given a value X and notes $\{1, 2, 7, 10\}$, find the minimum number of notes to create value X. You can use each note as many times as you want.

Greedy does not always work

Problem 1: Given a value X and notes $\{1, 2, 5, 10, 20, 50, 100\}$, find the minimum number of notes to create value X. You can use each note as many times as you want.

Answer: Greedy approach works. Pick largest note that is at most X and subtract from X. Repeat until value becomes 0.
E.g., for X=1477, you need fourteen 100s, one 50, one 20, one 5 and one 2.

Problem 2: Given a value X and notes $\{1, 2, 7, 10\}$, find the minimum number of notes to create value X. You can use each note as many times as you want.

Answer: Greedy approach does not work as before.
E.g., for X=14, you need two 7s, but greedy will give one 10, two 2s.

Greedy does not always work

Problem 1: Given a value X and notes $\{1, 2, 5, 10, 20, 50, 100\}$, find the minimum number of notes to create value X. You can use each note as many times as you want.

Answer: Greedy approach works. Pick largest note that is at most X and subtract from X. Repeat until value becomes 0.
E.g., for X=1477, you need fourteen 100s, one 50, one 20, one 5 and one 2.

Problem 2: Given a value X and notes $\{1, 2, 7, 10\}$, find the minimum number of notes to create value X. You can use each note as many times as you want.

Answer: Greedy approach does not work as before.
E.g., for X=14, you need two 7s, but greedy will give one 10, two 2s.

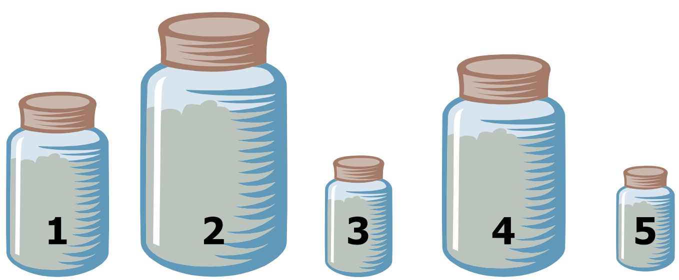
Greedy does not work always

Fractional Knapsack

Problem: A set of n items, with each item i having positive weight w_i and positive value v_i. You are asked to choose items with maximum total value so that the total weight is at most W. We are allowed to take fractional amounts (some percentage of each item).

Example:

Items:



Weight:	4 ml	8 ml	2 ml	6 ml	1 ml
Value:	\$12	\$32	\$40	\$30	\$50
Value: (\$ per ml)	\$3	\$4	\$20	\$5	\$50

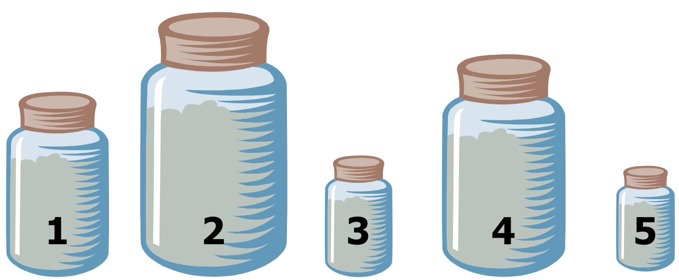
"knapsack" with 10ml

Fractional Knapsack

Problem: A set of n items, with each item i having positive weight w_i and positive value v_i. You are asked to choose items with **maximum total value** so that the **total weight** is **at most** W. We are allowed to take **fractional amounts** (some percentage of each item).

Example:

Items:



Weight:	4 ml	8 ml	2 ml	6 ml	1 ml
Value:	\$12	\$32	\$40	\$30	\$50
Value: (\$ per ml)	\$3	\$4	\$20	\$5	\$50

"knapsack" with 10ml

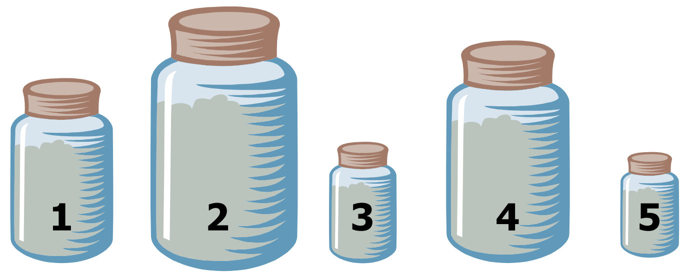
Solution:
- 1 ml of 5
- 2 ml of 3
- 6 ml of 4
- 1 ml of 2

Total Value: \$124

Fractional Knapsack

Idea: Greedy approach. Keep taking item with highest value to weight ratio until knapsack is full or run out of items.

Items:



Weight:	4 ml	8 ml	2 ml	6 ml	1 ml
Value:	\$12	\$32	\$40	\$30	\$50
Value:	\$3	\$4	\$20	\$5	\$50

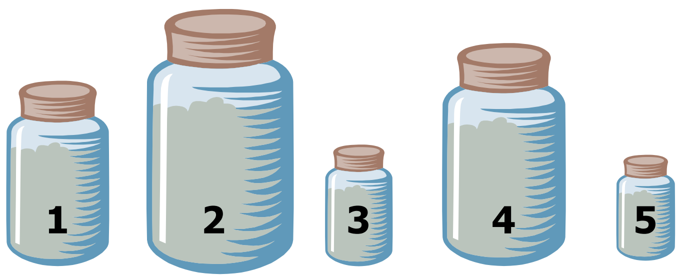
$W = 10$ ml

$value = \$0$

Fractional Knapsack

Idea: Greedy approach. Keep taking item with highest value to weight ratio until knapsack is full or run out of items.

Items:



Weight:	4 ml	8 ml	2 ml	6 ml	**1 ml**
Value:	\$12	\$32	\$40	\$30	\$50
Value:	\$3	\$4	\$20	\$5	**\$50**

$W = 10$ ml

$value = \$0$

Fractional Knapsack

Idea: Greedy approach. Keep taking item with highest value to weight ratio until knapsack is full or run out of items.

Items:

Weight:	4 ml	8 ml	2 ml	6 ml
Value:	\$12	\$32	\$40	\$30
Value:	\$3	\$4	\$20	\$5

$W = 9$ ml

$value = \$50$

Fractional Knapsack

Idea: Greedy approach. Keep taking item with highest value to weight ratio until knapsack is full or run out of items.

Items:

Weight:	4 ml	8 ml	**2 ml**	6 ml
Value:	\$12	\$32	\$40	\$30
Value:	\$3	\$4	**\$20**	\$5

$W = 9$ ml

$value = \$50$

Fractional Knapsack

Idea: Greedy approach. Keep taking item with highest value to weight ratio until knapsack is full or run out of items.

Items:

Weight:	4 ml	8 ml	6 ml
Value:	\$12	\$32	\$30
Value:	\$3	\$4	\$5

$W = 7$ ml

$value = \$90$

Fractional Knapsack

Idea: Greedy approach. Keep taking item with highest value to weight ratio until knapsack is full or run out of items.

Items:			
Weight:	4 ml	8 ml	**6 ml**
Value:	\$12	\$32	\$30
Value:	\$3	\$4	**\$5**

W = 7 ml

$value$ = \$90

Fractional Knapsack

Idea: Greedy approach. Keep taking item with highest value to weight ratio until knapsack is full or run out of items.

Items: 

Weight:	4 ml	8 ml
Value:	\$12	\$32
Value:	\$3	\$4

$W = 1$ ml

$value = \$120$

Fractional Knapsack

Idea: Greedy approach. Keep taking item with highest value to weight ratio until knapsack is full or run out of items.

Items:

Weight:	4 ml	8 ml
Value:	\$12	\$32
Value:	\$3	**\$4**

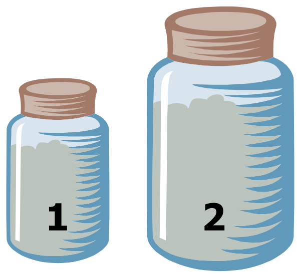
$W = 1$ ml

$value = \$120$

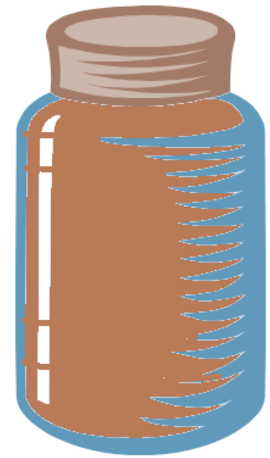
Fractional Knapsack

Idea: Greedy approach. Keep taking item with highest value to weight ratio until knapsack is full or run out of items.

Items:



Weight:	4 ml	7 ml
Value:	\$12	\$32
Value:	\$3	**\$4**



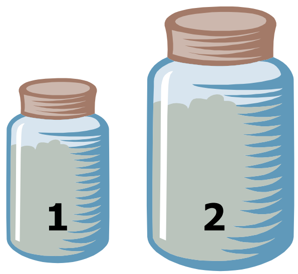
$W = 0$ ml

$value = \$124$

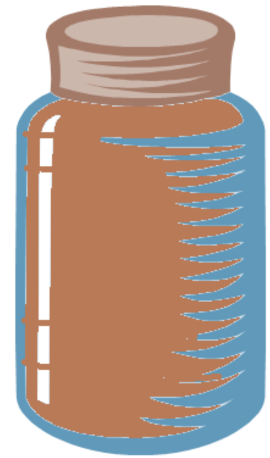
Running time: ?

Fractional Knapsack

Idea: Greedy approach. Keep taking item with highest value to weight ratio until knapsack is full or run out of items.

Items: 

Weight:	4 ml	7 ml
Value:	\$12	\$32
Value:	\$3	**\$4**



$W = 0$ ml

$value = \$124$

Running time: If we sort the items with respect to value to weight ratio then $\Theta(n \log n)$.

Fractional Knapsack

Pseudocode:

Items with $v[], w[]$, knapsack with W

For $i = 1$ to n **do**

  $\mathrm{r}[i] \leftarrow \frac{v[i]}{w[i]}$

$w \leftarrow 0$

$val \leftarrow 0$

While $w < W$ **do**

  Remove item i with highest $r[i]$

  If $w + w_i \leq W$ **then**

    $w \leftarrow w + w_i$

    $val \leftarrow val + v[i]$

  Else

    $w \leftarrow W$, $val \leftarrow val + (W - w) \cdot r[i]$

return val

Fractional Knapsack

Pseudocode:

Items with $v[], w[]$, knapsack with W

For $i = 1$ to n **do**

$\quad \mathrm{r}[i] \leftarrow \frac{v[i]}{w[i]}$

$w \leftarrow 0$

$val \leftarrow 0$

While $w < W$ **do**

$\quad$ **Remove** item i with highest $r[i]$

$\quad$ **If** $w + w_i \leq W$ **then**

$\quad\quad w \leftarrow w + w_i$

$\quad\quad val \leftarrow val + v[i]$

$\quad$ **Else**

$\quad\quad w \leftarrow W, val \leftarrow val + (W - w) \cdot r[i]$

return val

Compute the ratios

Initialization

While knapsack not full

If whole item fits

Fractional Knapsack

Pseudocode:

Items with $v[], w[]$, knapsack with W

For $i = 1$ to n **do**

$\quad r[i] \leftarrow \frac{v[i]}{w[i]}$

$w \leftarrow 0$

$val \leftarrow 0$

While $w < W$ **do**

$\quad$ **Remove** item i with highest $r[i]$

$\quad$ **If** $w + w_i \leq W$ **then**

$\qquad w \leftarrow w + w_i$

$\qquad val \leftarrow val + v[i]$

$\quad$ **Else**

$\qquad w \leftarrow W,\ val \leftarrow val + (W - w) \cdot r[i]$

return val

Compute the ratios

Initialization

While knapsack not full

If whole item fits

Percentage of item i that fits

$(W - w) \cdot r[i] = \frac{W-w}{w(i)} \cdot v(i)$

Fractional Knapsack

Pseudocode:

Items with $v[], w[]$, knapsack with W

For $i = 1$ to n **do**

$\quad r[i] \leftarrow \frac{v[i]}{w[i]}$

$w \leftarrow 0$

$val \leftarrow 0$

Sort $r[1], \ldots, r[n]$

While $w < W$ **do**

$\quad$ **Remove** item i with highest $r[i]$

$\quad$ **If** $w + w_i \leq W$ **then**

$\quad\quad w \leftarrow w + w_i$

$\quad\quad val \leftarrow val + v[i]$

$\quad$ **Else**

$\quad\quad w \leftarrow W,\ val \leftarrow val + (W - w) \cdot r[i]$

return val

This is fast, in $O(1)$ time.

Fractional Knapsack

Why greedy works: General argument. Suppose there is a better solution. Assume items are order in decreasing order of value per weight, i.e., $r_1 \geq r_2 \ldots \geq r_n$.

- Let $x_1, \ldots, x_n$ be the weight values of the items in the knapsack for the better solution.

Fractional Knapsack

Why greedy works: General argument. Suppose there is a better solution. Assume items are order in decreasing order of value per weight, i.e., $r_1 \geq r_2 \ldots \geq r_n$.

- Let $x_1, \ldots, x_n$ be the weight values of the items in the knapsack for the better solution.
- Since it is different from what greedy returns, there must be indices i, j so that $r_i > r_j$ and $x_j > 0$ and $x_i < w_i$.

Fractional Knapsack

Why greedy works: General argument. Suppose there is a better solution. Assume items are order in decreasing order of value per weight, i.e., $r_1 \geq r_2 \ldots \geq r_n$.

- Let $x_1, \ldots, x_n$ be the weight values of the items in the knapsack for the better solution.
- Since it is different from what greedy returns, there must be indices i, j so that $r_i > r_j$ and $x_j > 0$ and $x_i < w_i$.

value per weight of item i is larger than j

Fractional Knapsack

Why greedy works: General argument. Suppose there is a better solution. Assume items are order in decreasing order of value per weight, i.e., $r_1 \geq r_2 \ldots \geq r_n$.

- Let $x_1, \ldots, x_n$ be the weight values of the items in the knapsack for the better solution.
- Since it is different from what greedy returns, there must be indices i, j so that $r_i > r_j$ and $x_j > 0$ and $x_i < w_i$.

value per weight of item i is larger than j

Part or all of item j is in the knapsack

Fractional Knapsack

Why greedy works: General argument. Suppose there is a better solution. Assume items are order in decreasing order of value per weight, i.e., $r_1 \geq r_2 \ldots \geq r_n$.

- Let $x_1, \ldots, x_n$ be the weight values of the items in the knapsack for the better solution.
- Since it is different from what greedy returns, there must be indices i, j so that $r_i > r_j$ and $x_j > 0$ and $x_i < w_i$.

value per weight of item i is larger than j

Part or all of item j is in the knapsack

Not all of item i is in the knapsack

Fractional Knapsack

Why greedy works: General argument. Suppose there is a better solution. Assume items are order in decreasing order of value per weight, i.e., $r_1 \geq r_2 \ldots \geq r_n$.

- Let $x_1, \ldots, x_n$ be the weight values of the items in the knapsack for the better solution.
- Since it is different from what greedy returns, there must be indices i, j so that $r_i > r_j$ and $x_j > 0$ and $x_i < w_i$.
- Exchange part of item i, with part of item j. How much?

Say the minimum of $w_i - x_i \; and \; x_j$.

Fractional Knapsack

Why greedy works: General argument. Suppose there is a better solution. Assume items are order in decreasing order of value per weight, i.e., $r_1 \geq r_2 \ldots \geq r_n$.

- Let $x_1, \ldots, x_n$ be the weight values of the items in the knapsack for the better solution.
- Since it is different from what greedy returns, there must be indices i, j so that $r_i > r_j$ and $x_j > 0$ and $x_i < w_i$.
- Exchange part of item i, with part of item j. How much?

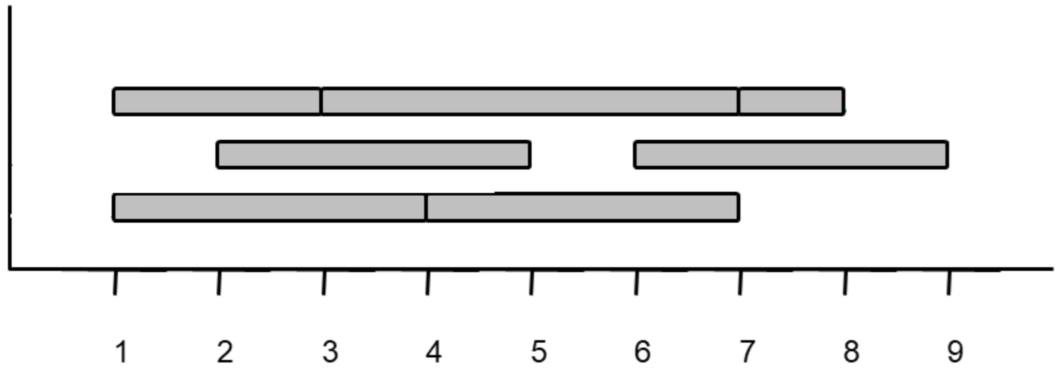
Say the minimum of $w_i - x_i$ *and* x_j.

Total value will increase by $(r_i - r_j) \cdot \min(w_i - x_i, x_j)$

Task scheduling

Problem: Given: a set T of n tasks, each having a start time s_i and a finish time f_i (where $s_i < f_i$)
Goal: Perform all the tasks using a minimum number of machines.
A machine can serve one task at a given time.

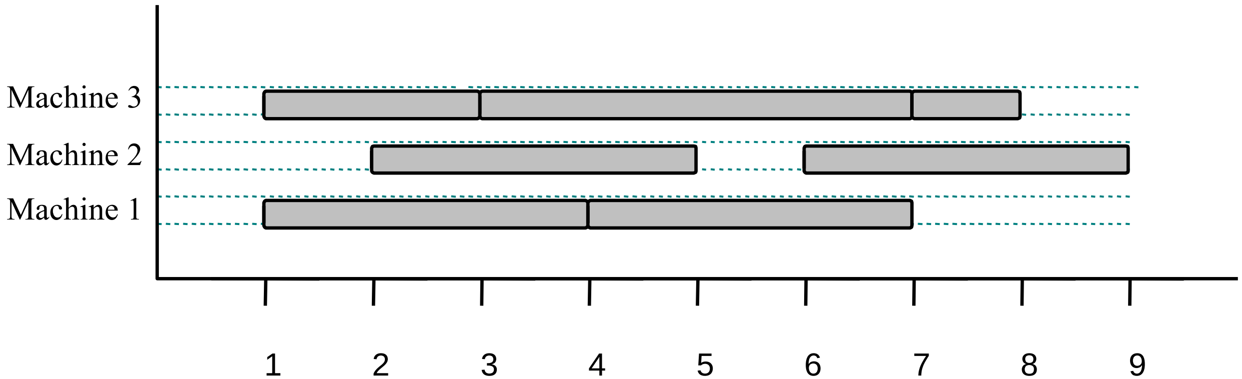
Example: 7 Tasks, [1,4], [1,3], [2,5], [3,7], [4,7], [6,9], [7,8]



Task scheduling

Problem: Given: a set T of n tasks, each having a start time s_i and a finish time f_i (where $s_i < f_i$)
Goal: Perform all the tasks using a minimum number of machines.
A machine can serve one task at a given time.

Example: 7 Tasks, [1,4], [1,3], [2,5], [3,7], [4,7], [6,9], [7,8]



Task scheduling

Idea: Greedy approach. Consider tasks in increasing order of their start time. Assign first task to machine 1 and set $K = 1$.
When considering a new task, if all machines are busy, create a new machine, set $K = K + 1$ and assign the new task to the new machine otherwise assign the new task to an available machine.

Example: 7 Tasks, [1,4], [1,3], [2,5], [3,7], [4,7], [6,9], [7,8]

Task scheduling

Idea: Greedy approach. Consider tasks in increasing order of their start time. Assign first task to machine 1 and set $K = 1$.
When considering a new task, if all machines are busy, create a new machine, set $K = K + 1$ and assign the new task to the new machine otherwise assign the new task to an available machine.

Example: 7 Tasks, [1,4], [1,3], [2,5], [3,7], [4,7], [6,9], [7,8]

$K = 1$

Machine 1 [1,4]

Task scheduling

Idea: Greedy approach. Consider tasks in increasing order of their start time. Assign first task to machine 1 and set $K = 1$.
When considering a new task, if all machines are busy, create a new machine, set $K = K + 1$ and assign the new task to the new machine otherwise assign the new task to an available machine.

Example: 7 Tasks, [1,4], [1,3], [2,5], [3,7], [4,7], [6,9], [7,8]

$K = 2$

Machine 1 [1,4]

Machine 2 [1,3]

Task scheduling

Idea: Greedy approach. Consider tasks in increasing order of their start time. Assign first task to machine 1 and set $K = 1$.
When considering a new task, if all machines are busy, create a new machine, set $K = K + 1$ and assign the new task to the new machine otherwise assign the new task to an available machine.

Example: 7 Tasks, [1,4], [1,3], [2,5], [3,7], [4,7], [6,9], [7,8]

$K = 3$

Machine 1 [1,4]

Machine 2 [1,3]

Machine 3 [2,5]

Task scheduling

Idea: Greedy approach. Consider tasks in increasing order of their start time. Assign first task to machine 1 and set $K = 1$.
When considering a new task, if all machines are busy, create a new machine, set $K = K + 1$ and assign the new task to the new machine otherwise assign the new task to an available machine.

Example: 7 Tasks, [1,4], [1,3], [2,5], [3,7], [4,7], [6,9], [7,8]

$K = 3$

Machine 1	[1,4]	
Machine 2	[1,3]	[3,7]
Machine 3	[2,5]	

Task scheduling

Idea: Greedy approach. Consider tasks in increasing order of their start time. Assign first task to machine 1 and set $K = 1$.
When considering a new task, if all machines are busy, create a new machine, set $K = K + 1$ and assign the new task to the new machine otherwise assign the new task to an available machine.

Example: 7 Tasks, [1,4], [1,3], [2,5], [3,7], [4,7], [6,9], [7,8]

$K = 3$

Machine 1	[1,4]	[4,7]
Machine 2	[1,3]	[3,7]
Machine 3	[2,5]	

Task scheduling

Idea: Greedy approach. Consider tasks in increasing order of their start time. Assign first task to machine 1 and set $K = 1$.
When considering a new task, if all machines are busy, create a new machine, set $K = K + 1$ and assign the new task to the new machine otherwise assign the new task to an available machine.

Example: 7 Tasks, [1,4], [1,3], [2,5], [3,7], [4,7], [6,9], [7,8]

$K = 3$

Machine 1	[1,4]	[4,7]
Machine 2	[1,3]	[3,7]
Machine 3	[2,5]	[6,9]

Task scheduling

Idea: Greedy approach. Consider tasks in increasing order of their start time. Assign first task to machine 1 and set $K = 1$.
When considering a new task, if all machines are busy, create a new machine, set $K = K + 1$ and assign the new task to the new machine otherwise assign the new task to an available machine.

Example: 7 Tasks, [1,4], [1,3], [2,5], [3,7], [4,7], [6,9], [7,8]

$K = 3$

Machine 1	[1,4]	[4,7]	[7,8]
Machine 2	[1,3]	[3,7]	
Machine 3	[2,5]	[6,9]	

Task scheduling

Idea: Greedy approach. Consider tasks in increasing order of their start time. Assign first task to machine 1 and set $K = 1$.
When considering a new task, if all machines are busy, create a new machine, set $K = K + 1$ and assign the new task to the new machine otherwise assign the new task to an available machine.

Why greedy works: General argument. Suppose there is a better solution, using $k - 1$ machines instead of k.

Task scheduling

Idea: Greedy approach. Consider tasks in increasing order of their start time. Assign first task to machine 1 and set $K = 1$.
When considering a new task, if all machines are busy, create a new machine, set $K = K + 1$ and assign the new task to the new machine otherwise assign the new task to an available machine.

Why greedy works: General argument. Suppose there is a better solution, using $k - 1$ machines instead of k.

- Let i be the first task that used Machine k. At that moment, there are must be $k - 1$ conflicting tasks with task i.

Task scheduling

Idea: Greedy approach. Consider tasks in increasing order of their start time. Assign first task to machine 1 and set $K = 1$.
When considering a new task, if all machines are busy, create a new machine, set $K = K + 1$ and assign the new task to the new machine otherwise assign the new task to an available machine.

Why greedy works: General argument. Suppose there is a better solution, using $k - 1$ machines instead of k.

- Let i be the first task that used Machine k. At that moment,

  there are must be $k - 1$ conflicting tasks with task i.

- All these $k - 1$ tasks have finishing times larger than s_i and starting times less than or equal to s_i.

Task scheduling

Idea: Greedy approach. Consider tasks in increasing order of their start time. Assign first task to machine 1 and set $K = 1$.
When considering a new task, if all machines are busy, create a new machine, set $K = K + 1$ and assign the new task to the new machine otherwise assign the new task to an available machine.

Why greedy works: General argument. Suppose there is a better solution, using $k - 1$ machines instead of k.

- Let i be the first task that used Machine k. At that moment,

  there are must be $k - 1$ conflicting tasks with task i.

- All these $k - 1$ tasks have finishing times larger than s_i and starting times less than or equal to s_i. These tasks are conflict with each other!

- So we have k tasks that conflict with each other, we need k machines!

Task scheduling

Idea: Greedy approach. Consider tasks in increasing order of their start time. Assign first task to machine 1 and set $K = 1$.
When considering a new task, if all machines are busy, create a new machine, set $K = K + 1$ and assign the new task to the new machine otherwise assign the new task to an available machine.

Why greedy works: General argument. Suppose there is a better solution, using $k - 1$ machines instead of k.

- Let i be the first task that used Machine k. At that moment,

  there are must be $k - 1$ conflicting tasks with task i.

- All these $k - 1$ tasks have finishing times larger than s_i and starting times less than or equal to s_i. These tasks are conflict with each other!

- So we have k tasks that conflict with each other, we need k machines! Contradiction!